# Intrusion-Tolerant Broadcast and Agreement Abstractions in the Presence of Byzantine Processes

Achour Mostéfaoui[†] Michel Raynal[⋆,‡]
[†] LINA, Université de Nantes, 44322 Nantes Cedex, France
[⋆] Institut Universitaire de France
[‡] IRISA, Université de Rennes 35042 Rennes Cedex, France
Achour.Mostefaoui@univ-nantes.fr raynal@irisa.fr

**Abstract**

A process commits a Byzantine failure when its behavior does not comply with the algorithm it is assumed to execute. Considering asynchronous message-passing systems, this paper presents distributed abstractions, and associated algorithms, that allow non-faulty processes to correctly cooperate, despite the uncertainty created by the net effect of asynchrony and Byzantine failures. These abstractions are broadcast abstractions (namely, no-duplicity broadcast, reliable broadcast, and validated broadcast), and agreement abstraction (namely, consensus).

While no-duplicity broadcast and reliable broadcast are well-known one-to-all communication abstractions, validated broadcast is a new all-to-all communication abstraction designed to address agreement problems. After having introduced these abstractions, the paper presents an algorithm implementing validated broadcast on top of reliable broadcast. Then the paper presents two consensus algorithms, which are reductions of multivalued consensus to binary consensus. The first one is a generic algorithm, that can be instantiated with unreliable broadcast or no-duplicity broadcast, while the second is a consensus algorithm based on validated broadcast. Finally, a third algorithm is presented that solves the binary consensus problem. This algorithm is a randomized algorithm based on validated broadcast and a common coin. The presentation of all the abstractions and their algorithms is done incrementally.

**Keywords**: Abstraction level, Agreement, Asynchronous message-passing system, Broadcast abstraction, Byzantine process, Common coin, Consensus, Distributed algorithm, Fault-tolerance, Intrusion-tolerance, Message validation, Reliable broadcast, Signature-free algorithm.

## 1 Introduction

**Distributed computing** Distributed computing is the part of computing that considers that (a) the computing entities (usually called nodes, processors, processes, agents, sensors, etc.) are geographically distinct, and (b) each entity has only a partial knowledge of the many parameters involved in the problem to be solved [1, 15, 24, 25, 26]. In the following, we use the term *process* to denote any computing entity. From an operational point of view this means that the processes of a distributed system need to exchange information, and agree in some way or another, in order to cooperate to a common goal. When designing a distributed algorithm, a main difficulty comes from the fact that no process can capture instantaneously the global state of the application it is a part of. As a simple example, it is impossible to distinguish a crashed process from a very slow process in an asynchronous system prone to process crashes. Said another way, a central issue of distributed computing consists in coping with the uncertainty created by asynchrony and

failures. As in sequential computing, a simple approach to facilitate the design of distributed applications consists in designing appropriate abstractions. With such abstractions, the application designer can think about solutions to solve problems at a higher conceptual level than the basic send/receive communication level.

**Byzantine failure** This failure type has first been introduced in the context of synchronous distributed systems [13, 22, 25], and then investigated in the context of asynchronous distributed systems [1, 15, 24]. A process has a *Byzantine* behavior when it arbitrarily deviates from its intended behavior (Byzantine failure) [7, 13]. Otherwise it is non-faulty. Let us notice that process crashes (unexpected halting) define a strict subset of Byzantine failures. An example of a bad behavior consists for a process to send different messages to distinct subset of processes, while it is assumed to send the same message to all[1].

**Communication and agreement abstractions** One of the most important communication abstractions encountered in fault-tolerant distributed computing is *Reliable Broadcast* [1, 5, 6, 8, 15, 24, 27]. Roughly speaking, reliable broadcast allows processes to broadcast messages, in such a way that all the non-faulty processes eventually deliver the same set of messages, and this set includes *all* the messages they have broadcast plus a subset of messages broadcast by faulty processes. *Consensus* is the most important agreement abstraction of fault-tolerant distributed computing [11]. Assuming each process proposes a value, it allows the non-faulty processes to agree on the same value, which has to satisfy some validity condition depending on both the proposed values and the failure model [15, 25].

**Byzantine-tolerant broadcast abstractions** This paper presents communication and agreement abstractions suited to distributed systems made up of $n$ processes, and where up to $t$ processes may exhibit Byzantine failures.

As far as communication is concerned, three abstractions are presented. The first two, which have been introduced in [3, 28], are one-to-all broadcast abstractions. (Albeit not new, we present them for completeness in order to have a self-contained paper. This provides us with a global view, which favors understanding and makes reading easier.) Both these abstractions ensure that a message broadcast by a non-faulty process is delivered by all the non-faulty processes. They differ in their requirement on the message broadcast by faulty processes. More precisely, we have the following.

- No-duplicity broadcast (ND-broadcast). As far as a message broadcast by a Byzantine process $p$ is concerned, ND-broadcast ensures that no two non-faulty processes deliver different messages from process $p$ [28]. Let us observe that this delivery rule allows some non-faulty processes to deliver the same message $m$ from the faulty process $p$, while others deliver no message from $p$.

- Reliable broadcast (RB-broadcast). This abstraction is stronger than the previous one. As far as a message broadcast by a Byzantine process $p$ is concerned, the RB-broadcast abstraction ensures that all the non-faulty processes deliver the same message from $p$, or none of them delivers a message from $p$ [3]. This is an "all-or-none" delivery rule.

[1] As noticed in [13], when designing Byzantine-tolerant algorithms, we are not concerned by the cause of the bad behavior. This bad behavior can be intentional (malicious) or simply the result of a transient fault that altered the local state of a process, thereby modifying its behavior in an unpredictable way. Some intentional bad behaviors are security flaws, which can be detected with appropriate security techniques. These security issues are out the scope of the paper.

Let us notice that, in both previous broadcast abstractions, if a message is delivered from a faulty process $p$, there is no requirement on its content. If the sender is correct, the content of a message that is delivered is the one that should be sent; if the sender is faulty, the content of the message can be arbitrary.

While the previous communication abstractions are one-to-all abstractions, the third abstraction (called VB-broadcast) is an all-to-all communication abstraction.

- Validated broadcast (VB-broadcast). Each process is assumed to broadcast a message. For a message to be delivered by a non-faulty process, its content needs to be *validated* [19]. A message content is validated by its sender as soon as it knows that at least one non-faulty process broadcast a message with the same content. As a process does not know if it is faulty or not (e.g., while it executes correctly its algorithm, a process may crash unexpectedly), a message content is validated by its sender as soon as it has received messages with the very same content from $(t + 1)$ distinct processes (such a set contains at least one non-faulty process). If the message broadcast by a process cannot be validated, the default value $\perp$ is delivered instead of it.

Each of these three broadcast abstractions requires $t < n/3$, and are consequently resilience-optimal (with respect to the maximal number of processes that can be faulty) [3, 24, 28]. Moreover, (as we will see) an algorithm implementing RB-broadcast can be obtained by using a relatively simple "echo" mechanism [3, 27, 28], and an algorithm implementing each of the $n$ internal broadcasts of a VB-broadcast instance can be obtained from two instances the RB-broadcast abstraction.[2]

**Content of the paper: Byzantine-tolerant agreement abstraction** The most important agreement abstraction encountered in distributed systems prone to process failures is consensus ([1, 5, 15, 26], to cite only books). It is well-known that consensus is impossible to solve in the basic asynchronous message-passing system model prone to even a single process crash failure [11]. This means that solving consensus despite both asynchrony and Byzantine processes requires to enrich the system with additional computational power. We consider here that this additional power is given by an underlying algorithm solving the binary consensus problem (consensus instance where the only values that can be proposed are $0$ and $1$). Such an algorithm (in short BBC, for Binary Byzantine Consensus) is described in the paper. This algorithm assumes that the processes can access an oracle called *common coin*, which outputs random numbers [23]. It is signature-free, resilience-optimal ($t < n/3$), and its expected number rounds is four.

The paper presents three multivalued Byzantine consensus algorithms (i.e., consensus algorithms where the set of values is not restricted to the values $0$ and $1$, and processes can be Byzantine). These algorithms, whose constructions are highly modular, are based on the previous broadcast abstractions and (as announced) a BBC algorithm. More precisely, The first two multivalued Byzantine consensus algorithms are obtained from a generic algorithm whose genericity parameter is a broadcast abstraction (namely, UB-broadcast –which captures unreliable broadcast–, and ND-broadcast). The third algorithm is based on the VB-broadcast abstraction. Interestingly, all these algorithms are signature-free, i.e., no underlying cryptography mechanism is used.

These Byzantine consensus algorithms differ in their cost (number and size of messages they use), and their requirement on $t$, from a "lower bound on $t$" (or resilience) point of view. The instance of the generic

[2] As already indicated, differently from validated broadcast, no-duplicity broadcast and reliable broadcast are not new. We present them in this paper to have a self-contained paper, which facilitates understanding and provides us with a global picture of broadcast abstractions.

Byzantine consensus algorithm based on UB-broadcast requires $t < n/5$, while its instance based on ND-broadcast requires $t < n/4$. Finally, the Byzantine consensus algorithm based on VB-broadcast requires $t < n/3$, and is consequently resilience-optimal.

**Intrusion-tolerant Byzantine consensus** The previous algorithms have a noteworthy property, namely, if the faulty processes collude and propose the very same value $v$, while no non-faulty process proposes it, then $v$ cannot be decided. This property is called *intrusion-tolerance* in [19]. More generally, when the most proposed value is proposed by too few non-faulty processes, a default value denoted $\perp$ may be decided.

**Road map** The paper is made up of 8 sections. Section 2 presents the computation model. Then Section 3 presents the broadcast abstractions which have been previously sketched, and algorithms implementing them. Section 4 presents the intrusion-tolerant Byzantine consensus problem. Then, sections 5 and 6 present a suite of intrusion-tolerant multivalued Byzantine consensus algorithms that differ mainly in the underlying broadcast abstraction they use. Section 7 presents a new binary consensus algorithm based on random bits and the VB-broadcast abstraction. Finally, Section 8 concludes the paper.

# 2 Computation Model

**Asynchronous processes** The system is made up of a finite set $\Pi$ of $n > 1$ asynchronous sequential processes, namely $\Pi = \{p_1, \ldots, p_n\}$. "Asynchronous" means that each process proceeds at its own speed, which can vary arbitrarily with time, and remains unknown to the other processes.

**Communication network** The processes communicate by exchanging messages through an asynchronous reliable point-to-point network. "Asynchronous" means that a message that has been sent is received by its destination process after a finite time , i.e., there is no finite bound on message transfer delays. "Reliable" means that the network neither losses, duplicates, modifies, nor creates messages. "Point-to-point" means that there is a bi-directional communication channel between each pair of processes. Hence, when a process receives a message, it can identify its sender.

A process $p_i$ sends a message to a process $p_j$ by invoking the primitive "send TYPE$(m)$ to $p_j$", where TYPE is the type of the message and $m$ its content. To simplify the presentation, it is assumed that a process can send messages to itself. A process receives a message by executing "receive()".

**Failure model** Up to $t$ processes can exhibit a *Byzantine* behavior. A Byzantine process is a process that behaves arbitrarily: it can crash, fail to send or receive messages, send arbitrary messages, start in an arbitrary state, perform arbitrary state transitions, etc. Hence, a Byzantine process, which is assumed to send a message $m$ to all the processes, can send a message $m_1$ to some processes, a different message $m_2$ to another subset of processes, and no message at all to the other processes. Byzantine processes can collude to "pollute" the computation. Let us notice that, as each pair of processes is connected by a channel, no Byzantine process can impersonate a correct process (otherwise, no non-trivial computation can be done). Moreover, it is assumed that the Byzantine processes do not control the network.

**Terminology** A process that exhibits a Byzantine behavior is called *faulty*. Otherwise, it is *non-faulty*. Given an execution, $\mathcal{C}$ denotes the set of processes that are non-faulty in that execution, and $\mathcal{F}$ denotes the set of faulty processes.

**Multiset** Distributed algorithms presented in the paper use multisets. A *multiset* (also called *bag*) differs from a set in that it can contain several copies of the same value. Given a multiset $rec_i$ managed by a process $p_i$, the operation $\#\mathsf{equal}(v, rec_i)$ issued by $p_i$ denotes the number of occurrences of $v$ in $rec_i$, while $\#\mathsf{differ}(v, rec_i)$ denotes the number of occurrences of values different from $v$ in $rec_i$, namely, $\#\mathsf{differ}(v, rec_i) = |rec_i| - \#\mathsf{equal}(v, rec_i)$.

**Notation** This system model is denoted $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\emptyset]$. In the following, this model is enriched with a constraint on $t$ and a specific broadcast abstraction. As an example, $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/5, \text{RB}]$ is $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\emptyset]$ in which at most $t < n/5$ processes are faulty, and processes communicate using the RB-broadcast abstraction.

**Remark** Making realistic assumptions on the reliability of the asynchronous network, and the fact that Byzantine processes cannot impersonate a correct process, are requirements on the computation model built by the underlying system layer [15, 24, 26]. The model parameter $t$ states that the algorithms are guaranteed to work only when there are at most $t$ Byzantine processes.

# 3 Broadcast Abstractions

This section defines the broadcast abstractions sketched in the Introduction, and presents algorithms implementing each of them. The first two, ND-broadcast and RB-broadcast, are from [28] and [3], respectively.

All these broadcast abstractions are implemented from the basic send/receive network primitives, which means that, while they provide us with distinct abstraction levels, they do not provide processes with additional computing power.

**Notation** When considering the broadcast abstraction XX (where XX stands for UB, ND, RB, or VB),we say that a process "XX-broadcasts" or "XX-delivers" a message.

**Unreliable broadcast** The simple broadcast (UB-broadcast) is defined by a pair of operations denoted $\mathsf{UB\_broadcast}()$ and $\mathsf{UB\_deliver}()$. $\mathsf{UB\_broadcast}$ $\text{TYPE}(m)$ is used as a shortcut for

**for each** $j \in \{1, \ldots, n\}$ $\mathsf{send}$ $\text{TYPE}(m)$ $\mathsf{to}$ $p_j$ **end for**,

and $\mathsf{UB\_deliver}()$ is synonym with $\mathsf{receive}()$. This means that a message UB-broadcast by a non-faulty process is UB-delivered at least by all the non-faulty processes. Differently, while it is assumed to send the same message to all the processes, a faulty process can actually send different messages to distinct processes and no message to others. Hence the name "unreliable broadcast" (sometimes also called "best effort broadcast"). Trivially, an invocation of $\mathsf{UB\_broadcast}$ $\text{TYPE}(m)$ costs one communication step and $n$ messages. The corresponding system model is denoted $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\text{UB}]$.

**Remark** When measuring the cost of a broadcast abstraction we do not take into account the size of the "data message" that is broadcast. This is because this size is application-dependent. Consequently, we only consider the size of the additional control information required by the corresponding broadcast implementation.

**The no-duplication property** The definition of each XX-broadcast abstraction includes the following no-duplication property: a non-faulty process $p_i$ XX-delivers at most one message from any process $p_j$. This property states that the corresponding XX-broadcast abstraction is not allowed to create message duplicates. As this property follows trivially from the implementation of each broadcast abstraction, it is no longer mentioned in the following.

**A preliminary lemma** The following lemma is used in the correctness proof of the algorithms implementing the ND-Broadcast and RB-broadcast abstractions (Theorem 1 and Theorem 2). It is stated here to help better understand the description of the corresponding algorithms.

**Lemma 1.** *Let $t < n/3$. We have*
(a) $n - t > \frac{n+t}{2}$,
(b) *any set containing more than $\frac{n+t}{2}$ distinct processes, contains at least $(t+1)$ non-faulty processes, and*
(c) *any two sets $Q_1$ and $Q_2$ of at least $\lfloor \frac{n+t}{2} \rfloor + 1$ processes have at least one correct process in their intersection.*

**Proof** Proof of (a). $n > 3t \Leftrightarrow 2n > n + 3t \Leftrightarrow 2n - 2t > n + t \Leftrightarrow n - t > \frac{n+t}{2}$.
Proof of (b). We have $\frac{n+t}{2} \geq \frac{4t+1}{2} = 2t + \frac{1}{2}$, from which it follows that any set of more than $\frac{n+t}{2}$ distinct processes contains at least $2t + 1$ processes. The proof then follows from the fact that any set of $2t + 1$ distinct processes contains at least $t + 1$ non-faulty processes.
Proof of (c). Notice that $(x > \frac{n+t}{2}) \equiv (x \geq \lfloor \frac{n+t}{2} \rfloor + 1)$.

- $Q_1 \cup Q_2 \subseteq \{p_1, \ldots, p_n\}$. Hence, $|Q_1 \cup Q_2| \leq n$.
- $|Q_1 \cap Q_2| = |Q_1| + |Q_2| - |Q_1 \cup Q_2| \geq |Q_1| + |Q_2| - n \geq 2(\lfloor \frac{n+t}{2} \rfloor + 1) - n > 2(\frac{n+t}{2}) - n = t$. Hence, $|Q_1 \cap Q_2| \geq t + 1$, from which it follows that $Q_1 \cap Q_2$ contains at least one correct process.

$\square_{Lemma\ 1}$

## 3.1 The no-duplicity broadcast abstraction

**No-duplicity broadcast** The ND-broadcast communication abstraction, introduced by Toueg [28], is defined by the operations ND_broadcast() and ND_deliver(), which provide the processes with a higher abstraction level than UB-broadcast but do not add computational power ($\mathcal{BZ}\_\mathcal{AS}_{n,t}$[UB] and $\mathcal{BZ}\_\mathcal{AS}_{n,t}$[ND] have the same computational power, namely the same power as $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\emptyset]$).

Considering an instance of ND-broadcast where ND_broadcast() is invoked by a process $p_i$, this communication abstraction is defined by the following properties.

- ND-Validity. If a non-faulty process ND-delivers a message from a correct process $p_i$, then $p_i$ invoked ND-broadcast.
- ND-no-duplicity. No two non-faulty processes ND-deliver distinct messages from $p_i$.

- ND-Termination-1. If the sender $p_i$ is non-faulty, all the non-faulty processes eventually ND-deliver its message.

Let us observe that, if the sender $p_i$ is faulty, it is possible that some non-faulty processes ND-deliver a message from $p_i$ while others do not. The no-duplicity property prevents the non-faulty processes from ND-delivering different messages from a faulty sender.

**An algorithm implementing ND-broadcast** Assuming $t < n/3$, Algorithm 1 (from [28]) implements the ND-broadcast abstraction. It is shown in [28] that $t < n/3$ is an upper bound on the model parameter $t$ when one has to implement ND-broadcast in an asynchronous message-passing system prone to process Byzantine failures.

```
operation ND_broadcast MSG(v_i) is
(01)  UB_broadcast INIT(i, v_i).

when INIT(j, v) is UB_delivered do
(02)  if (first UB_delivery of INIT(j, −))
(03)     then UB_broadcast ECHO(j, v)
(04)  end if.

when ECHO(j, v) is UB_delivered do
(05)  if (ECHO(j, v) UB_delivered from more than (n+t)/2
          different processes and MSG(j, v) not yet ND_delivered)
(06)     then ND_deliver MSG(j, v)
(07)  end if.
```

Algorithm 1: Implementing ND-broadcast ($t < n/3$) [28]

The algorithm considers that a process is allowed to ND-broadcast only one message. Adding sequence numbers allows processes to ND-broadcast several messages. In that case, the process identity associated with each message has to be replaced by a pair made up of a sequence number and a process identity.

When a process $p_i$ wants to ND-broadcast a message whose content is $v_i$, it UB-broadcasts the message $\text{INIT}(i, v_i)$ (line 01). When a process $p_i$ receives (UB-delivers) a message $\text{INIT}(j, -)$ for the first time, it UB-broadcasts a message $\text{ECHO}(j, v)$ where $v$ is the data content of the $\text{INIT}()$ message (line 02). If the message $\text{INIT}(j, v)$ received is not the first message $\text{INIT}(j, -)$, $p_j$ is Byzantine and the message is discarded. Finally, when $p_i$ has received the same message $\text{ECHO}(j, v)$ from more than $(n + t)/2$ processes, it locally ND-delivers $\text{MSG}(j, v)$ (lines 05-06).

**Theorem 1.** *Algorithm* 1 *implements the ND-broadcast abstraction in the system model* $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/3, \text{UB}]$. (Proof in [20, 28].)

It is easy to see that this implementation uses two consecutive communication steps and $n(n + 1)$ underlying messages ($n$ in the first communication step, and $n^2$ in the second one). Moreover, the size of the control information added to a message is $\log_2 n$ (sender identity).

**Remark** Let us notice that replacing at line 06 "more than $\frac{n+t}{2}$ different processes" by "$(n - t)$ different processes" leaves the algorithm correct. As $n - t > \frac{n+t}{2}$ (Item (a) of Lemma 1), it follows that using "more

than $\frac{n+t}{2}$ different processes" provides a weaker ND-delivery condition than "$(n-t)$ different processes", and consequently a more efficient algorithm from a "message ND-delivery" point of view. As a simple numerical example, considering $n = 21$ and $t = 2$, we have $n - t = 19$, which is much greater than the required value 12 ($> \frac{n+t}{2} = 11.5$).

### 3.2 The reliable broadcast abstraction

**Reliable broadcast** The RB-broadcast abstraction has been proposed by G. Bracha [3]. It is proved in [4] that $t < n/3$ is an upper bound on $t$ when one has to implement such an abstraction. RB-broadcast provides the processes with the operations RB_broadcast() and RB_deliver() defined by the following properties.

- RB-Validity. If a non-faulty process RB-delivers a message from a correct process $p_i$, then $p_i$ invoked the operation RB_broadcast().
- RB-no-duplicity. No two non-faulty processes ND-deliver distinct messages from $p_i$.
- RB-Termination-1. If the sender $p_i$ is non-faulty, all the non-faulty processes eventually ND-deliver its message.
- RB-Termination-2. If a non-faulty process RB-delivers a message from $p_i$ (possibly faulty) then all the non-faulty processes eventually RB-deliver the same message from $p_i$.

RB-broadcast is ND-broadcast plus the RB-Termination-2 property. Hence, not only two non-faulty processes cannot RB-deliver different messages from a given process, but it is no longer possible that one of them RB-delivers a message while the other does not. $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\text{RB}]$ and $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\emptyset]$ have the same computational power.

**An algorithm implementing RB-broadcast** Algorithm 2, which assumes $t < n/3$, implements RB-broadcast. It is a simple variant of an algorithm proposed in [3]. It is presented here in an incremental way from the previous ND-broadcast algorithm.

While ND-broadcast Algorithm 1 requires two sequential communications steps (message INIT() followed by messages ECHO()), the implementation of RB-broadcast requires three consecutive communications steps: message INIT(), followed by messages ECHO(), followed by messages READY().

The first seven lines of the algorithm are similar to the corresponding lines of the ND-broadcast algorithm. The only difference lies in the lines 05-06, where the ND-delivery is replaced by the UR-broadcast of the message READY$(j, v)$.

The aim of the last step of the algorithm (lines 08-13) is to ensure that all or none of the non-faulty processes RB-deliver the message MSG$(j, v)$ from $p_j$. To that end, the RB-delivery predicate requires that $p_i$ UB-delivers $(2t+1)$ copies of READY$(j, v)$, which means at least $(t+1)$ copies from non-faulty processes (line 11).

**Theorem 2.** *Algorithm 2 implements the RB-broadcast abstraction in the system model* $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/3, \text{UB}]$. (Proof in [3, 20].)

As we have seen, this algorithm uses three consecutive communication steps and $n(2n + 1)$ underlying messages ($n$ in the first communication step, and $n^2$ in each of the second and third steps). Moreover, the size of the control information added to a message is $\log_2 n$ (sender identity).

```
operation RB_broadcast MSG(v_i) is
(01) UB_broadcast INIT(i, v_i).

when INIT(j, v) is UB_delivered do
(02) if (first UB_delivery of INIT(j, −))
(03)    then UB_broadcast ECHO(j, v)
(04) end if.

when ECHO(j, v) is UB_delivered do
(05) if (ECHO(j, v) UB_delivered from more than (n+t)/2
         different processes and READY(j, v) not yet UB_broadcast)
(06)    then UB_broadcast READY(j, v)
(07) end if.

when READY(j, v) is UB_delivered do
(08) if (READY(j, v) UB_delivered from (t + 1) different processes
         and READY(j, v) not yet UB_broadcast)
(09)    then UB_broadcast READY(j, v)
(10) end if;
(11) if (READY(j, v) UB_delivered from (2t + 1) different processes
         and MSG(j, v) not yet RB_delivered)
(12)    then RB_deliver MSG(j, v)
(13) end if.
```

Algorithm 2: Implementing RB-broadcast ($t < n/3$) [3]

### 3.3 The validated broadcast abstraction

**Validated broadcast** The VB-broadcast communication abstraction is an *all-to-all* communication abstraction designed to be used in the implementation of distributed agreement abstractions. VB-broadcast integrates a notion of message *validation*, namely, assuming that each non-faulty process VB-broadcasts a message, it requires that, for a message to be VB-delivered, its content $v$ be validated; otherwise the default value $\perp$ is VB-delivered instead of it. For a message with content $v$ to be valid, a message with the same content $v$ has to be VB-broadcast by at least one non-faulty process. As no process $p_i$ knows if it is itself faulty or non-faulty (e.g., if a process executes correctly its algorithm and then unexpectedly crashes, it is faulty), for a message $m$ to be valid in the presence of up to $t$ faulty processes, messages with the same content need to be VB-broadcast by "enough" processes, where "enough" means "at least $(t + 1)$" (including its sender $p_i$). As already indicated, if a message is not validated, the default value $\perp$ is delivered instead of it.

VB-broadcast provides the processes with two operations denoted VB_broadcast() and VB_deliver(). In a VB-broadcast instance each non-faulty process invokes VB_broadcast() once, and VB-delivers messages from at least $(n - t)$ distinct processes. The content of a message that is VB-delivered can be a value that has been VB-broadcast or the default value $\perp$. VB-broadcast is defined by the following properties.

- VB-Validity. As previously, this property relates the outputs (VB-delivered messages) to the inputs (VB-broadcast messages). It is made up of two sub-properties.
  - VB-Justification. Let $p_i$ be a non-faulty process that VB-delivers a message $m$ as the value VB-broadcast by some (faulty or non-faulty) process. If $m \neq \perp$, there is at least one non-faulty

process that invoked VB_broadcast MSG$(m)$.

- VB-Obligation. If all the non-faulty processes VB_broadcast the same value $v$, each non-faulty process VB-delivers $m = v$ from each non-faulty process.

- VB-Uniformity. If a non-faulty process VB-delivers a message from $p_i$ (possibly faulty), all the non-faulty processes eventually VB-deliver the same message from $p_i$ (which can be a validated non-$\perp$ value or the default value $\perp$).

- VB-Termination. If $p_i$ is non-faulty and VB-broadcasts $m$, all the non-faulty processes eventually VB-deliver the same message $m'$, where $m'$ is $m$ or $\perp$.

### 3.4 An algorithm implementing VB-broadcast

Assuming $t < n/3$, Algorithm 3 implements the all-to-all VB-broadcast abstraction. Let us recall that all-to-all means here that all the non-faulty processes are assumed to invoke VB_broadcast(). As already said, a process VB-delivers at least $n - t$ messages. This implementation uses consecutively two RB-broadcast abstractions. It is made up of two parts.

```
operation VB_broadcast(v_i) is
(01) RB_broadcast INIT(i, v_i);
(02) let rec_i = multiset of values RB_delivered to p_i;
(03) wait until (|rec_i| ≥ n − t);
(04) if (#equal(v_i, rec_i) ≥ n − 2t) then aux_i ← "yes"
(05)    else aux_i ← "no"
(06) end if;
(07) RB_broadcast VALID(i, aux_i).

for 1 ≤ j ≤ n VB-delivery background task T_i[j]:
(08) wait until (VALID(j, x) and INIT(j, v) RB_delivered from p_j);
(09) if (x = "yes") then wait (#equal(v, rec_i) ≥ n − 2t); d ← v
(10)                 else wait (#differ(v, rec_i) ≥ t + 1); d ← ⊥
(11) end if;
(12) VB_deliver(d) at p_i as the value VB-broadcast by p_j.
```

Algorithm 3: VB-broadcast on top of reliable broadcast ($t < n/3$, code of $p_i$)

In the first part, a process $p_i$ first invokes RB_broadcast INIT$(i, v_i)$ and waits until it has RB-delivered messages from at least $n - t$ processes (lines 01-03). The values RB-delivered are deposited in a multiset denoted $rec_i$. Then, if value $v_i$ has been RB-delivered from at least $n - 2t \geq t + 1$ processes (which means that it was RB-broadcast by at least one non-faulty process), $p_i$ validates it by assigning "yes" to $aux_i$. Otherwise $p_i$ sets $aux_i$ to "no" at line 05 (in this case, $v_i$ is not validated). Then, $p_i$ issues a second RB-broadcast (line 07) to disseminate $aux_i$ to all processes.

The second part is made up of $n$ tasks, which execute in the background. The task $T_i[j]$ is associated with the VB-delivery of the message from $p_j$. It starts by the wait statement for both the value $v$ RB-broadcast by $p_j$ and the value $x$ RB-broadcast also by $p_j$ ($x$ indicates the validation status attached to $v$ by its sender $p_j$). Let us remember that each time a message INIT$(-, w)$ is RB-delivered to $p_i$, the corresponding value $w$ is added to $rec_i$, which means that, after the predicate $|rec_i| \geq n - t$ has become true at line 03, the set $rec_i$ still keeps on being updated when new messages INIT() are RB-delivered to $p_i$.

- If $x$ = "yes", as $p_j$ can be Byzantine, $v$ was not necessarily validated by a non-faulty process. Hence, $p_i$ has to check it. To that end, $p_i$ waits until the predicate $\#\mathsf{equal}(v, rec_i) \geq n - 2t$ becomes true (line 09). When this predicate becomes true (if ever it does), it follows from $n - 2t \geq t + 1$ that $\#\mathsf{equal}(v, rec_i) \geq t + 1$. If this occurs, $v$ is VB-delivered to $p_i$ as being the value VB-broadcast by $p_j$.

- Similarly, if $x$ = "no", $p_i$ waits until $rec_i$ contains more than $t$ occurrences of values different from $v$ (the value RB-delivered from $p_j$), which means that at least one non-faulty process did not validate $v$. When this occurs (if ever it does, line 10), $p_i$ VB-delivers $\perp$ as the value VB-broadcast by $p_j$.

It is possible that the waiting predicate used at line 09 or line 10 never becomes satisfied. When this occurs, the corresponding sender process $p_j$ is necessarily a faulty process. The waiting condition becomes always satisfied when $p_j$ is a non-faulty process, and can become satisfied for some faulty senders $p_j$.

As two instances of RB-broadcast are used, the algorithm requires $2 \times 3 = 6$ communication steps, and as VB-broadcast is an all-to-all abstraction ($n$ invocations for each instance of RB-broadcast), the algorithm uses $2n^2(2n + 1)$ messages, that is $O(n^3)$.

**Theorem 3.** *Algorithm* 3 *implements the validated broadcast abstraction in the system model* $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/3, \mathrm{UB}]$.

**Proof** Proof of the VB-Termination property. This property states that, if a process $p_i$ is non-faulty and VB-broadcast $m$, then all the non-faulty processes eventually VB-deliver the same message $m'$ from $p_i$, where $m'$ is $m$ or $\perp$.

As there are at least $n - t$ non-faulty processes, and each of them VB-broadcasts a value, we eventually have $|rec_j| \geq n - t$ at every non-faulty process $p_j$. Hence, no non-faulty process can block forever at line 03 and will RB-broadcasts message VALID() at line 07. We now consider two cases.

- The non-faulty process $p_i$ RB-broadcasts VALID($i$,"yes"). It follows from line 09 that (a) $d = v_i$ (the value VB-broadcast by $p_i$), and (b) $rec_i$ contains at least $n - 2t$ copies of $v = v_i$, i.e., $p_i$ has RB-delivered messages INIT $(-, v)$ from $n - 2t$ different processes. Due to the RB-Uniformity of RB-broadcast, each non-faulty process $p_j$ eventually RB-delivers both these $n - 2t$ messages INIT $(-, v)$, and the message VALID($i$,"yes") from $p_i$. It follows that $p_j$ eventually VB-delivers $v = v_i$ at line 09.

- The non-faulty process $p_i$ RB-broadcasts VALID($i$,"no"). It follows from the termination property of RB-broadcast that each non-faulty process $p_j$ RB-delivers VALID($i$,"no") from $p_i$. Moreover, it follows from the test line 04 that, if $p_i$ RB-broadcast VALID($i$,"no"), that, among the $n - t$ values in $rec_i$, less then $n - 2t$ values are equal to $v_i$, i.e. more than $t$ values are different from $v_i$. Hence due to the RB-Uniformity property of RB-broadcast, every non-faulty process $p_j$ eventually RB-delivers at least $t + 1$ values different from $v_i$, and consequently VB-delivers $\perp$ at line 10.

Proof of the VB-Uniformity property. This property states that, if a non-faulty process $p_i$ VB-delivers a message from $p_j$ –possibly faulty–, then all the non-faulty processes eventually VB-deliver the same message from $p_j$. Let $p_i$ be a non-faulty process that VB-delivers a value $d$ from $p_j$. This means that $p_i$ has previously RB-delivered a message INIT$(j, v)$ and a message VALID$(j, x)$ from $p_j$ at the latest in its delivery task $T_i[j]$. The proof of this property is very similar to the previous one.

Hence, $p_i$ has RB-delivered (1) a message VALID$(j, x)$ and a message INIT$(j, v)$ from $p_j$, and (2) a multiset $rec_i$ of values that satisfies some property (depending on the value of $x$). As $p_i$ is non-faulty, it follows from the RB-Uniformity property of RB-broadcast, that every non-faulty process $p_k$ eventually RB-delivers (1) both VALID$(j, x)$ and INIT$(j, v)$, and (2) a multiset $rec_k$ of values such that eventually $rec_k = rec_i$. As the value $x$ RB-delivered to $p_i$ and $p_k$ is the same, it follows from the waiting condition (used at line 09 or line 10, according to the value of $x$) that $p_k$ eventually VB-delivers at line 12 the same value $d$ as $p_i$.

Proof of the VB-Obligation property. This property states that if all the non-faulty process VB-broadcast the same value $v$, each of them VB-delivers $v$ as the value VB-broadcast by each of them. As each non-faulty process $p_j$ VB-broadcasts the value $v$, it follows that it RB-broadcasts INIT$(j, v)$ (line 07). Consequently this value $v$ eventually appears at least $(n - 2t)$ times in the multiset $rec_i$ of every non-faulty process $p_i$. Hence, each non-faulty process $p_i$ VB-broadcasts the message VALID$(i,$"yes"$)$ and (from the RB-termination property) each non-faulty process $p_k$ RB-delivers the message VALID$(i,$"yes"$)$. Consequently, each non-faulty process $p_k$ executes the task $T_k[i]$ with respect to each non-faulty process $p_i$ (and possibly also with respect to faulty processes). The waiting predicate of line 09 is then eventually satisfied at $p_k$, and this is true for value $v$ only. When this occurs, each non-faulty process $p_k$ VB-delivers $v$ as the value VB-broadcast by the non-faulty process $p_i$.

Proof of the VB-Justification property. This property states that, if the VB-delivered value $m$ is such that $m \neq \perp$, there is at least one non-faulty process that invoked VB_broadcast MSG$(m)$. If $m \neq \perp$ is VB-delivered by a non-faulty process $p_i$ as the value VB-broadcast by $p_j$, this value appears at least $(n-2t)$ times in $rec_i$ (waiting predicate of line 09). As $n - 2t \geq t + 1$, it follows that at least one non-faulty process has VB-broadcast $m$. $\square_{Theorem\ 3}$

### 3.5 Comparing the broadcast abstractions

Table 1 compares the costs of the three previous broadcast abstractions. Considering one broadcast instance, the second column indicates the broadcast type (1-to-$n$ or $n$-to-$n$). The third column indicates the length of the longest sequence of causally-related messages generated by the corresponding algorithm. The fourth column presents the size of the additional control information that each message has to carry (the $\log_2 n$ comes from the fact that the identity of the process that broadcasts a message has to be sent together with it when forwarded by another process). The fifth column indicates the number of implementation messages used by the corresponding algorithms. Finally, the last column states the constraint on $t$ required to implement the abstraction in $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\emptyset]$.

| *broadcast* | *$x$-to-$y$ type* | *#com. steps* | *message size* | *#messages* | *resilience ($t$)* |
|---|---|---|---|---|---|
| UB | 1-to-$n$ | 1 | constant | $n$ | $n > t$ |
| ND | 1-to-$n$ | 2 | $\log_2 n$ | $n(n+1)$ | $n > 3t$ |
| RB | 1-to-$n$ | 3 | $\log_2 n$ | $n(2n+1)$ | $n > 3t$ |
| VB | $n$-to-$n$ | 6 | $\log_2 n$ | $2n^2(2n+1)$ | $n > 3t$ |

Table 1: Costs and constraints on the broadcast abstractions

# 4 Intrusion-Tolerant Byzantine Consensus and Underlying Enriched Model

## 4.1 Byzantine consensus

**Byzantine consensus** The consensus problem has been informally stated in the Introduction. Assuming that each non-faulty process proposes a value, each of them has to decide on a value in such a way that the following properties are satisfied.

- C-Termination. Every non-faulty process eventually decides on a value.
- C-Agreement. No two non-faulty processes decide on different values.
- C-Obligation (validity). If all the non-faulty processes propose the same value $v$, then $v$ is decided.

**Intrusion-tolerant Byzantine (ITB) consensus** In Byzantine consensus, if not all the non-faulty processes propose the same value, any value can be decided. As indicated in the Introduction, we are interested here in a more constrained version of the consensus problem in which a value proposed only by faulty processes cannot be decided. This consensus problem instance is defined by the C-Termination, C-Agreement and C-Obligation properties stated above plus the following C-Non-intrusion property (which is a validity property).

- C-Non-intrusion (validity). A decided value is a value proposed by a non-faulty process or $\perp$.

The fact that no value proposed only by faulty processes can be decided gives its name (namely *intrusion-tolerant*) to that consensus problem instance.

**Binary consensus** The consensus is *binary* when only two different values (e.g., $b$ and $\overline{b}$) can be proposed. Interestingly, binary Byzantine consensus has the following property.

**Property 1.** *The* ITB *binary consensus problem is such that it is always possible for a correct process to decide a value proposed by a correct process.*

**Proof** It follows from the C-obligation property that any correct process decides $b$ when all the correct processes proposed $b$. Otherwise, at least one correct process proposed $\overline{b}$, and then any value $b$ or $\overline{b}$ may be decided by a correct process without violating the C-Non-intrusion property. $\square_{Property\ 1}$

## 4.2 Enriched model for multivalued ITB consensus

**Additional power is required** It is well-known that Byzantine consensus cannot be solved when $t \geq n/3$ in synchronous systems [13, 22]. Moreover, consensus cannot be solved in asynchronous systems as soon as even only one process may crash [11], which means that Byzantine consensus cannot be solved either as soon as one process can be faulty. Said another way, additional computational power is needed if one wants to solve Byzantine consensus in an asynchronous system.

Such an additional power can be obtained by randomization (e.g., [2, 9, 12, 23, 28]), failure detectors (e.g., [18]), additional synchrony assumptions (e.g., [10, 14, 16, 17]), or even the assumption that there is a binary consensus algorithm that is given for free by the underlying system (e.g., [9, 21, 29]).

**Enriched model for multivalued ITB consensus** In the following, BBC denotes any algorithm that solves the Byzantine binary consensus problem. (Such algorithms are described in [3, 9, 12, 28]. A novel BBC algorithm based on the VB-broadcast abstraction is presented in Section 7). Let $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\text{XX}, \text{BBC}]$ denote the system model $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\emptyset]$ enriched with BBC (which adds computational power) and the broadcast abstraction XX (which provides a higher abstraction level than send/receive).

As announced in the Introduction, the aim is to design a generic multivalued ITB consensus algorithm on top of $\mathcal{BZ}\_\mathcal{AS}_{n,t}[\text{XX}, \text{BBC}]$.

# 5 Generic Consensus Based on the UB or ND-Broadcast Abstractions

This section presents a generic multivalued ITB consensus algorithm that can be instantiated with UB-broadcast or ND-broadcast. It uses two rounds for each process to compute a value it proposes to the underlying binary consensus. The instance based on UB-broadcast requires $t < n/5$, while the one based on ND-broadcast requires $t < n/4$.

## 5.1 Principles and description of the algorithm

Algorithm 4 is a generic algorithm. A process invokes propose$(v_i)$ where $v_i$ is the value it proposes to the consensus. It terminates when it executes the statement return() (line 17), which supplies it with the decided value. (In order to prevent confusion, the operation of the underlying binary consensus is denoted bin_propose().)

```
operation propose(v_i) is
(01) XX_broadcast EST1(v_i);
(02) wait until (EST1(−) messages XX_delivered from n − t proc.);
(03) let rec1_i = multiset of values carried by the EST1 messages;
(04) if (∃v : #equal(v, rec1_i) ≥ n − 2t)
(05)     then aux_i ← v else aux_i ← ⊥ end if;
(06) XX_broadcast EST2(aux_i);
(07) wait until (EST2(−) messages XX_delivered from n − t proc.);
(08) let rec2_i = multiset of values carried by the EST2 messages;
(09) if (∃v ≠ ⊥ : #equal(v, rec2_i) ≥ n − 2t)
(10)     then bp_i ← 1 else bp_i ← 0 end if;
(11) if (∃v ≠ ⊥ : v ∈ rec2_i)
(12)     then let v = most frequent non-⊥ value in rec2_i;
(13)          res_i ← v
(14)     else res_i ← ⊥
(15) end if;
(16) b_dec_i ← bin_propose(bp_i); % underlying BBC algorithm %
(17) if (b_dec_i = 1) then return(res_i) else return(⊥) end if.
```

Algorithm 4: Generic intrusion-tolerant Byzantine multivalued consensus

In order to reduce the Byzantine consensus problem to its binary counterpart to benefit from BBC, the processes first exchange the values they propose. If a process sees that a value $v$ has been proposed "enough" times, it proposes 1 to BBC, otherwise it proposes 0. Then, if 1 is decided from BBC, the non-faulty processes decide the value $v$ that has been proposed "enough" times, otherwise they decide $\perp$ (lines

11-17). For this to work, If a process $p_i$ proposes 1 to the underlying BBC algorithm because it has seen enough copies of a value $v$, it must be sure that any other non-faulty process $p_j$ will be able to decide $v$ even if it has proposed 0 to BBC (because it has not seen enough copies of $v$).

This issue is solved by two asynchronous rounds executed before invoking the underlying BBC algorithm (lines 01-15). The messages of the first round and the second round are typed EST1 and EST2, respectively. Interestingly, we will state below two properties $PR1$ and $PR2$ that are the same as the properties used in [18, 24] to solve consensus on top of an asynchronous system enriched with any of Chandra and Toueg's failure detectors [8].

It is important to remark that, at the abstraction level of the consensus algorithm, a message carries only a type (EST1 or EST2) and a proposed value or $\perp$. Hence, considering that proposed values have constant size, the size of the messages of the algorithm is $O(1)$ (no message is required to carry array-like data structures whose size would depend on $n$).

### 5.2 First additional round

The aim of this round (lines 01-05) is to direct each process $p_i$ to define a "new" proposed value $aux_i$ in such a way that the values $aux_i$ of the non-faulty processes satisfy the following property (Lemma 2):

$$\begin{array}{c} PR1 \equiv \\ \forall i, j \in \mathcal{C} : \big((aux_i \neq \perp) \wedge (aux_j \neq \perp)\big) \Rightarrow \\ [(aux_i = aux_j = v) \wedge \\ (v \text{ has been proposed by a non-faulty process})]. \end{array}$$

Hence this round replaces (for the non-faulty processes) the set of values they collectively propose by a non-empty set including at most two values (namely, a value $v$ proposed by a non-faulty process and $\perp$).

From an operational point of view, this is obtained as follows. The processes first exchange (with the help of the underlying broadcast facility) the values they propose (lines 01-02). The values delivered at $p_i$ are kept in the multiset $rec1_i$. Then, if there is a value $v$ in $rec1_i$ such that $\#\mathsf{equal}(v, rec1_i) \geq n - 2t$, $v$ is assigned to $aux_i$. Otherwise $aux_i = \perp$.

### 5.3 Second additional round

The aim of the second round (lines 06-15) is to establish the following property denoted $PR2$ (Lemma 3) in order the result of the underlying BBC algorithm can be safely exploited as described previously (lines 16-17). The local variable $bp_i$ contains the value proposed by $p_i$ to the underlying BBC algorithm, and $res_j$ contains the non-$\perp$ value that any non-faulty process $p_j$ will decide if the default value $\perp$ is not decided. Let us recall that $\mathcal{C}$ denotes the set of processes that are non-faulty in the considered execution.

$$\begin{array}{c} PR2 \equiv \\ (\exists i \in \mathcal{C} : bp_i = 1) \Rightarrow (\forall j \in \mathcal{C} : res_j = res_i = v \neq \perp). \end{array}$$

Operationally, this is obtained as follows. With the help of the underlying broadcast abstraction the non-faulty processes exchange the values of their $aux_i$ variables. The values delivered at $p_i$ are saved in the multiset $rec2_i$. (This multiset contains $n - t$ values, and, due to $PR1$, those can be $\perp$, a non-$\perp$ value $v$ proposed by a non-faulty process, and at most $t$ arbitrary values sent by faulty processes.)

If there is a non-$\perp$ value $v$ such that $\#\mathsf{equal}(v, rec2_i) \geq n - 2t$, $p_i$ proposes $bp_i = 1$ to the binary consensus BBC. Otherwise, $p_i$ has not seen enough copies of a value $v \neq \perp$ and consequently proposes $bp_i = 0$. In all cases, $p_i$ defines $res_i$ as the most frequent non-$\perp$ value it has received. As the proof of Lemma 3 will show, if a non-faulty process $p_i$ invokes $\mathsf{bin\_propose}(1)$, each non-faulty process will have the same non-$\perp$ value in its local variable $res_j$.

### 5.4 Proof of the algorithm

**Lemma 2.** *$PR1$ holds in both system models $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/5, \mathrm{UB}]$ and $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/4, \mathrm{ND}]$.*

**Proof** Let $p_i$ and $p_j$ be two non-faulty processes such that $aux_i = v \neq \perp$. We consider separately each case stated in the lemma assumption.

*Case 1*: $t < n/5$ and the non-faulty processes use the UB-broadcast abstraction.

As $aux_i = v \neq \perp$, it follows that $\#\mathsf{equal}(v, rec1_i) \geq n - 2t$ (line 04). Hence, due to the UB-broadcast, among the $n - t$ messages it has UB-delivered (from different processes), at least $n - 2t$ are $\mathrm{EST1}(v)$. As at most $t$ processes are faulty, it follows that at least $n - 3t$ non-faulty processes have UB-broadcast a message $\mathrm{EST1}(v)$. Consequently, at most $n - (n - 3t) = 3t$ processes may send $w \neq v$ to $p_j$. As $3t < n - 2t$, $p_j$ never assigns $w$ to $aux_j$.

Finally, the proof that $v$ has been proposed by a non-faulty process follows from the observation that $v$ has been sent by at least $n - 2t > t$ non-faulty processes.

*Case 2*: $t < n/4$ and the non-faulty processes use the ND-broadcast.

In that case, $p_i$ has ND-delivered at least $(n - 2t)$ messages $\mathrm{EST1}(v)$, from different processes, and $p_j$ has ND-delivered at least $n - 2t$ messages $\mathrm{EST1}(w)$ from different processes. As $n > 4t$, it follows that there is a process $p_x$ such that $p_i$ has ND-delivered $\mathrm{EST1}(v)$ from $p_x$ and $p_j$ has ND-delivered $\mathrm{EST1}(w)$ from $p_x$. But, be $p_x$ faulty or non-faulty, this is impossible due to the ND-duplicity property (if a non-faulty process ND-delivers a value from a process $p_x$, any other non-faulty process either ND-delivers the same value from $p_x$ or does not ND-deliver a message from $p_x$). It follows that we have $v = w$.

Finally, similarly to the previous case, the proof that $v$ has been proposed by a non-faulty process follows from the observation that $v$ has been ND-broadcast by at least $n - 2t > t$ non-faulty processes. $\square_{Lemma\ 2}$

**Lemma 3.** *$PR2$ holds in both system models $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/5, \mathrm{UB}]$ and $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/4, \mathrm{ND}]$.*

**Proof** Let $p_i$ be a process such that $bp_i = 1$. It follows from lines 07-10 that the multiset $rec2_i$ contains $n-t$ values (including $\perp$). From lines 09-10, we also have $(bp_i = 1) \Rightarrow \big(\exists v \neq \perp : \#\mathsf{equal}(v, rec2_i) \geq n-2t\big)$, from which we conclude that $p_i$ has delivered at least $n-2t$ messages $\mathrm{EST2}(v)$. Moreover, due to Lemma 2, the values sent by non-faulty processes are only $v$ or $\perp$. Let us consider separately each case stated in the lemma assumption.

*Case 1*: $t < n/5$ and the non-faulty processes use UB-broadcast. As there are at most $t$ faulty processes, at most $t$ messages $\mathrm{EST2}(v)$ UB-delivered by $p_i$ are from faulty processes. Consequently, at least $n - 3t$ non-faulty processes have UB-broadcast $\mathrm{EST2}(v)$ to $p_j$. As $p_j$ waits for $n - t$ messages, it can miss at most $t$ messages $\mathrm{EST2}(v)$ from non-faulty processes (this is because, in the worst case, the $t$ messages missed by $p_j$ are from non-faulty processes that UB-broadcast $\mathrm{EST2}(v)$). Consequently, $p_j$ UB-delivers at least $n - 4t$ messages $\mathrm{EST2}(v)$ from non-faulty processes. As $n > 5t$, we have $\#\mathsf{equal}(v, rec2_j) > n - 4t \geq t + 1$. Let

us finally notice that, as at most $t$ processes are faulty, $p_j$ UB-delivers at most $t$ messages EST2$(-)$ carrying values different from $v$ and $\perp$. Hence, $\forall w \neq \perp$ we have $\#\mathsf{equal}(v, rec2_j) > t \geq \#\mathsf{equal}(w, rec2_j)$, which proves the lemma.

*Case 2*: $t < n/4$ and the non-faulty processes use ND-broadcast. In that case, due to ND-broadcast, no two non-faulty processes can ND-deliver different values from the same faulty process. The worst case is then when (a) $t$ processes are faulty and ND-broadcast the same value $w \notin \{v, \perp\}$, and (b) $p_j$ ND-delivers these $t$ messages EST2$(w)$. We trivially have $t \geq \#\mathsf{equal}(w, rec2_j)$. On another side, as $\#\mathsf{equal}(v, rec2_i) \geq n - 2t \geq 2t + 1$, and $p_j$ misses at most $t$ messages EST2$(v)$, we have $\#\mathsf{equal}(v, rec2_j) \geq t + 1$. Hence, we have $\#\mathsf{equal}(v, rec2_j) > t \geq \#\mathsf{equal}(w, rec2_j)$, which concludes the proof of the lemma. $\square_{Lemma\ 3}$

**Theorem 4.** *Algorithm* 4 *solves the* ITB *multivalued consensus problem in both* $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/5, \mathrm{UB}, \mathrm{BBC}]$ *and* $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/4, \mathrm{ND}, \mathrm{BBC}]$. *The termination property is the one inherited from the underlying* BBC *algorithm (deterministic:* C-Termination*), or randomized:* see Section 7.1).

**Proof** Proof of the Termination property (every non-faulty process decides). As at most $t$ processes are faulty, no non-faulty process blocks forever at line 02 or line 07. Due to the termination property of the underlying binary consensus algorithm BBC, every non-faulty process decides.

Proof of the C-Agreement property (no two non-faulty processes decide differently). If BBC returns 0, all the non-faulty processes decide $\perp$, and C-Agreement trivially follows. Hence, let us consider that BBC returns 1. It then follows from Property 1 of BBC that there is a non-faulty process $p_i$ such that $bp_i = 1$. Hence, due to Lemma 3, any non-faulty process $p_j$ is such that $res_j = v$, and all the non-faulty processes decide $v$.

Proof of the C-Obligation property (if all the non-faulty processes propose the same value, that value is decided). Let us assume that all the non-faulty processes propose value $v$. Let $p_i$ be any non-faulty process. We then have $\#\mathsf{equal}(v, rec1_i) \geq n - 2t$ at each non-faulty process $p_i$, and consequently each of the (at least) $n - t$ non-faulty process sends EST2$(v)$ (line 06). So, each non-faulty process delivers at least $n - 2t$ of these messages and we have $\#\mathsf{equal}(v, rec2_i) \geq n - 2t$. Hence, any non-faulty $p_i$ is such that $bp_i = 1$ and sets $res_i$ to $v$. Due to the C-obligation property of BBC algorithm, value 1 is decided, and consequently all the non-faulty processes decide $v$.

Proof of the C-Non-intrusion property (a non-$\perp$ value proposed only by faulty processes cannot be decided). If a non-$\perp$ value is decided, it follows from Property 1 of the underlying BBC that a non-faulty process $p_i$ has proposed 1. Hence, we have $bp_i = 1$, and consequently $\#\mathsf{equal}(v, rec2_i) \geq n - 2t$. As there are at most $t$ faulty processes, it follows that non-faulty processes have broadcast EST2$(v)$, which in turn implies that $n - 2t$ processes have broadcast EST1$(v)$, i.e., at least $n - 3t \geq t + 1$ processes have broadcast EST1$(v)$, from which we finally conclude that $v$ has been proposed by non-faulty processes. $\square_{Theorem\ 4}$

# 6 A Consensus Algorithm Based on the VB-Broadcast Abstraction

This section presents an intrusion-tolerant Byzantine consensus algorithm based on the VB-broadcast abstraction. Algorithm 5 requires $t < n/3$ and has consequently an optimal resilience. It requires a single round (instead of two as in Algorithm 4). As it is based on VB-broadcast, this round requires six communication steps.

*Principles and description of the algorithm* After it has VB-broadcast its value, a process $p_i$ waits for EST() messages from $n-t$ processes and deposits the corresponding values in the multiset $rec_i$. Then, $p_i$ checks if (1) (in addition to $\perp$) it has VB-delivered exactly one non-$\perp$ value $v$, and (2) $v$ has been VB-broadcast by at least $n-2t$ processes (line 04). If there is such a value, $p_i$ proposes 1 to the underlying binary consensus, otherwise it proposes 0 (line 05).

Finally, $p_i$ decides $\perp$ if the underlying binary consensus BBC returns 0 (lines 11). Differently, if 1 is returned, $p_i$ waits until it has VB-delivered $(n-2t)$ messages EST() carrying the very same value $v$ (line 09) and then decides that value (line 10). Let us notice that, among these $(n-2t)$ messages, some have been already VB-delivered at line 02. The important point is (as shown in the proof) that the net effect of (a) the VB-broadcast, (b) the predicate used at line 04, and (c) the predicate used in the wait statement at line 09, ensures that if a non-faulty process invokes bin_propose(1), then all the non-faulty processes eventually VB-deliver $(n-2t)$ times the same value $v$ and decide it.

```
operation propose(v_i) is
(01) VB_broadcast EST(v_i);
(02) wait until (EST(−) messages VB_delivered from n − t proc.);
(03) let rec_i = multiset of values v s.t. EST(v) is VB_delivered to p_i;
(04) if (∃v ≠ ⊥ : #equal(v, rec_i) ≥ n − 2t)
        ∧ (rec_i contains a single non-⊥ value)
(05)    then bp_i ← 1 else bp_i ← 0
(06) end if;
(07) b_dec_i ← bin_propose(bp_i);  % underlying BBC consensus %
(08) if (b_dec_i = 1)
(09)    then wait until (∃v ≠ ⊥ : #equal(v, rec_i) ≥ n − 2t);
(10)         return(v)
(11)    else return(⊥)
(12) end if.
```

Algorithm 5: Intrusion-tolerant Byzantine consensus algorithm based on VB-broadcast ($t < n/3$)

**On the predicate "$rec_i$ contains a single non-$\perp$ value" used at line 04** The aim of this predicate is to ensure that, if $bp_i = bp_j = 1$ (where $p_i$ and $p_j$ are two non-faulty processes), then the multisets $rec_i$ and $rec_j$ contain only instances of thesame value $v$ (plus possibly instances of $\perp$).

To motivate this predicate, let us consider that the predicate of line 04 is restricted to its first part, namely, "$\exists\, v :\ \#\mathsf{equal}(v, rec_i) \geq n-2t$". Assuming $n = 10$ and $t = 3$, let us consider the case where, at line 01, four processes VB-broadcast the message EST($v$), while six processes VB-broadcast the message EST($w$). Moreover, let us consider the following execution:

- On the one side, $p_i$ VB-delivers $n - t = 7$ messages EST(), four that carry $v$ and three that carry $w$. As $\#\mathsf{equal}(v, rec_i) = 4 \geq n - 2t = 4$, the restricted predicate is satisfied for $v$, and $p_i$ assigns 1 to $bp_i$.
- On the other side, $p_j$ VB-delivers $n - t = 7$ messages EST(), four that carry $w$ and three that carry $v$. As $\#\mathsf{equal}(w, rec_i) = 4 \geq n - 2t = 4$, the restricted predicate is satisfied for $w$, and $p_j$ assigns 1 to $bp_i$.

It follows that we have $bp_i = bp_j = 1$ ($p_i$ and $p_j$ being non-faulty processes), while $v$ is the value that will be decided by $p_i$ if the underlying BBC algorithm returns 1, and the value decided by $p_j$ will be $w \neq v$. It is easy to see that the second part of the predicate of line 04 prevents this bad scenario from occurring.

**Theorem 5.** *Algorithm* 5 *solves the* ITB *multivalued consensus problem in the system model* $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/3, \mathrm{VB}, \mathrm{BBC}]$. *As in Theorem 4, the termination property is the one inherited from the underlying* BBC *algorithm.*

**Proof** Proof of the C-Termination property (every non-faulty process decides). If the underlying BBC algorithm returns 0, termination is trivial. Hence, let us consider that 1 is returned. Due to Property 1 of BBC, there is a non-faulty process $p_i$ such that $bp_i = 1$, which in turn implies that, at line 02, $p_i$ has received at least $(n - 2t)$ messages $\mathrm{EST}(v)$. Due to the VB-Uniformity property of VB-broadcast, any non-faulty process eventually VB-delivers these $(n - 2t)$ messages $\mathrm{EST}(v)$. Hence, no non-faulty process $p_j$ blocks forever at line 09, which concludes the proof.
Proof the C-Agreement property (no two non-faulty processes decide differently). The proof is similar to the previous one. If BBC returns 0, agreement is trivial. If 1 is returned, it follows from $n - 2t > t$ and the fact that –at any non-faulty process $p_i$– there is no $w \neq v$ such that $w \in rec_i$ (second predicate of line 04), that the value $v$ the processes are waiting for at line 09 is unique, which completes the proof of the agreement property.

Proof of the C-Obligation property (if all the non-faulty processes propose the same value, that value is decided). If all the non-faulty processes propose the same value $v$, it follows from the VB-Obligation property that $v$ is necessarily validated, and from the VB-Termination property that all the non-faulty processes VB-deliver at least $(n - 2t)$ messages $\mathrm{EST}(v)$. Moreover, as $n - 2t > t$, there is a single such value $v$. Due to VB-Justification property, a value VB-broadcast only by faulty processes cannot be validated and consequently no non-faulty process can VB-deliver it. This means that only $v$, $\perp$ or nothing at all can be VB-delivered from a faulty process. It follows that, at each non-faulty process $p_i$, the predicate of line 04 is satisfied and $p_i$ proposes $bp_i = 1$. Due to the C-Obligation property of BBC, they all decide 1 and consequently decide the same proposed value $v$.

Proof of the C-Non-intrusion property (a non-$\perp$ value proposed only by faulty processes cannot be decided). If a value $w$ is proposed only by faulty processes, it follows from the VB-Justification property that no non-faulty process $p_i$ VB-delivers it. If the underlying BBC algorithm returns 0, $w$ is not decided. If BBC returns 1, we have seen in the proof of the C-Agreement property that the processes decide a value $v$ such that at least $(n - 2t)$ messages $\mathrm{EST}(v)$ have been VB-delivered. As $n - 2t > t$, it follows that $w$ cannot be decided. $\square_{Theorem\ 5}$

A discussion on additional properties of ITB consensus in deterministic vs non-deterministic scenarios can be found in [20].

# 7 A Randomized VB-Based Byzantine Binary Consensus Algorithm

This section presents a particularly simple randomized Byzantine binary consensus algorithm (that can be used as the underlying BBC algorithm on which rely the multivalued Byzantine consensus algorithms previously described). The additional power needed to solve consensus is given here by random coins. This algorithm, which is optimal from a resilience point of view ($t < n/3$), is based on the validated broadcast abstraction. More precisely, each round requires one VB-broadcast instance.

When looking at Byzantine consensus algorithms that are optimal from a resilience point of view (i.e., algorithms able to cope with up to $t < n/3$ faulty processes), the best consensus algorithm we are aware of has rounds made up of three communication steps [7]. Moreover, this algorithm is based on signatures (public key cryptography). As far as signature-free algorithms are concerned, the best resilience-optimal algorithm we are aware of, that uses control information whose size is only $O(\log_2 n)$, is the one described in [28], which requires five communication steps per round. Algorithm 6 is signature-free and requires only six communication steps per round.

### 7.1 Randomized model

**Common coin** The asynchronous system is equipped with a *common coin* as defined by Rabin [23], and improved in [7] in order to get rid of the trusted dealer. Such an oracle is denoted CC. The corresponding enriched –from a computational power point of view– system model is consequently denoted $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/3, \mathrm{CC}]$. A common coin can be seen as a global entity that delivers a sequence of random bits $b_1, b_2, \ldots, b_r, \ldots$ to processes (each bit $b_r$ has the value 0 or 1, with probability $1/2$).

More precisely, this oracle provides the processes with a primitive denoted $\mathsf{random}()$ that returns a bit each time it is called by a process. In addition to being random, this bit has the following global property: the $r$th invocation of $\mathsf{random}()$ by any non-faulty process $p_i$ returns it the bit $b_r$. This means that the $r$th invocations of $\mathsf{random}()$ by any pair of non-faulty processes $p_i$ and $p_j$ return them $b_r$, whatever the times at which each of these invocations occur. It is important to notice that the network has no access to the common coin (the reader interested in the implementation of a common coin can consult [7].)

**On randomized consensus** When using additional computing power provided by random coins, the consensus termination property can no longer be deterministic. *Randomized consensus* is defined by C-Validity (Obligation), C-Agreement, plus the following termination property [2, 23]: Every non-faulty process decides with probability 1. For round-based algorithms, this termination property can be re-stated as follows. For any non-faulty process $p_i$: $\lim_{r\to+\infty} \big(\text{Probability}[p_i \text{ decides by round } r]\big) = 1$.

### 7.2 The algorithm

*Underlying principles and description of the algorithm.* In Algorithm 6, a process $p_i$ invokes $\mathsf{bin\_propose}(v_i)$ where $v_i$ is the value it proposes. It decides when it executes the statement $\mathsf{decide}(v)$ (line 08). The design of this algorithm is close to an algorithm proposed in [12]. Its fundamental difference is that it is resilience-optimal ($t < n/3$), while the one described in [12] requires $t < n/5$.

The local variable $est_i$ of a process $p_i$ keeps its current estimate of the decision value (initially, $est_i = v_i$). The processes proceed by consecutive asynchronous rounds. Thus, the pair $(r_i, est_i)$ of a non-faulty process $p_i$ describes its current state ($r_i$ is $p_i$'s current round number). The first part of the algorithm (lines 01-04) is devoted to communication occurring during a round. The second part (lines 05-10) defines the management of the local estimate $est_i$ and the decision rule. There is one VB-broadcast instance per round. To distinguish the messages EST() associated with different VB-broadcast instances, these messages are tagged by their round number, namely EST$[r](v)$ denotes a round $r$ message carrying the value $v$. More precisely, we have the following.

- At every round $r_i$, each non-faulty process $p_i$ VB-broadcasts EST$[r_i](est_i)$, and waits until it has VB-delivered EST$[r_i](-)$ from at least $n - t$ processes (lines 02-04).

- In the second part, $p_i$ first computes the random number $s$ associated with the current round $r_i$ (line 05). Then, $p_i$ checks if (a) it has received a non-$\perp$ value $v$ from at least $n-2t$ different processes, and (b) $v$ is the only non-$\perp$ value in $rec_i$ (line 06). If this predicate holds, $p_i$ adopts $v$ as new estimate (line 07) and decides the random value $s$ if $v = s$ (line 08). If the predicate is false, $p_i$ updates its estimate $est_i$ to the random value $s$. In all cases, $p_i$ starts a new asynchronous round.

The statement decide() allows the invoking process to decide but does not stop its execution. Hence, a process executes rounds forever. This facilitates the description of the algorithm. Using techniques such as the one developed in [12] allows a process to both decide and stop.

**Remark** It is possible to add the following test after line 04:

**if** $(\exists v : \#\mathsf{equal}(v, rec_i) \geq n - t)$ **then** $\mathsf{decide}(v)$ **end if.**

This allows to always terminate in a single round whatever the value of the common coin when no process commits Byzantine failure and all processes propose the same value. (This scenario is likely to happen in actual executions.)

```
operation bin_propose(v_i) is
est_i ← v_i; r_i ← 0;
repeat forever
(01) r_i ← r_i + 1;
(02) VB_broadcast EST[r_i](est_i);
(03) let rec_i = multiset of values est
                 such that EST[r_i](est) has been VB_delivered to p_i;
(04) wait until (|rec_i| ≥ n − t);
(05) s_i ← random();
(06) if (∃v ≠ ⊥ : #equal(v, rec_i) ≥ n − 2t)
        ∧ (rec_i contains a single non-⊥ value)
(07)    then est_i ← v;
(08)     if (v = s) ∧ (p_i has not yet decided) then decide(v) end if
(09)    else est_i ← s
(10) end if
end repeat.
```

Algorithm 6: Randomized binary Byzantine consensus based on VB-broadcast ($t < n/3$)

## 7.3 Proof

**Lemma 4.** *Let $t < n/3$. Consider the situation where, at the beginning of a round $r$, all the non-faulty processes have the same estimate value $v$. These processes will never change their estimates, thereafter.*

**Proof** As all the non-faulty processes VB-broadcast the same value $v$ at the beginning of round $r$ (line 02), it follows from the VB-obligation property of VB-broadcast, that the only values that can be VB-delivered are $v$ (VB-broadcast by each of them and possibly from Byzantine processes) and $\perp$ (from Byzantine processes). Moreover, as each non-faulty process $p_i$ waits for $n - t$ messages (line 04), it will VB-deliver at least $n - 2t$ values $v$; as $n > 3t$, at most $t$ of them can be VB-broadcast by Byzantine processes (due to the VB-validity property, a value $w \neq v$ VB-broadcast by a Byzantine process $p_j$ cannot be validated, and consequently $\perp$ or no value at all is VB-delivered from such a process $p_j$). Hence, the predicate of line 06 is satisfied, and $p_i$ sets $est_i$ to $v$ (line 07), which concludes the proof of the lemma. $\square_{Lemma\ 4}$

Let $COND(v, i)$ be the predicate that $p_i$ tests at line 06.

**Lemma 5.** *Let $n > 3t$. If two non-faulty processes $p_i$ and $p_j$ are such that both $COND(v, i)$ and $COND(w, j)$ hold at round $r$, then $v = w$.*

**Proof** By the VB-Uniformity property of VB-broadcast, no two non-faulty processes VB-deliver different values from the same process. Hence, if $COND(v, i)$ holds for some non-faulty process $p_i$, no other non-faulty process $p_j$ can VB-deliver a value $w \neq v$ from the set of $(n-t)$ processes whose VB-broadcasts built the set $rec_i$. Consequently, if $p_j$ VB-delivers a value $w \neq v$, the number of occurrences of $w$ is necessarily at most $t < n - 2t$, and consequently $COND(w, j)$ cannot be satisfied. $\square_{Lemma\ 5}$

**Lemma 6.** *Let $t < n/3$. If all the non-faulty processes propose the same value $v$, then no value $v' \neq v$ can be decided.*

**Proof** This lemma is an immediate consequence of Lemma 4. As all estimates of the non-faulty processes remain equal to $v$, it follows from line 08 that no value $v' \neq v$ can be returned by a non-faulty process. $\square_{Lemma\ 6}$

**Lemma 7.** *No two non-faulty processes decide different values.*

**Proof** Let $r$ be the first round during which non-faulty processes decide. If two processes $p_i$ and $p_j$ decide at round $r$, they decide at line 08 the value $s$ computed by the common coin for that round. Moreover, before deciding during round $r$, a process updated its estimate to the decided value $s$. Hence, all processes that decide during round $r$ decide the same value $s$, and have their estimates equal to the decided value.

Let us now consider the case of a a processes $p_x$ for which, during round $r$, (a) the predicate of line 06 is satisfied (i.e., $COND(w, x)$ is true), (b) while the decision predicate at line 08 is not. As the predicate of line 06 is satisfied for both $p_x$ and any process $p_i$ that decides at line 08 (i.e., both $COND(w, x)$ and $COND(i, v)$ are true), it follows from Lemma 5 that $w = v$, which means that it is not possible that the decision predicate of $p_x$ be false. Hence, $p_x$ decides during round $r$, exactly as $p_i$.

Let us finally consider the case of a non-faulty process $p_k$ such that $COND(-, k)$ does not hold at line 06 during round $r$. It follows from line 09 that $p_k$ updates its estimate to the random value $s$ associated with round $r$. Hence, all such processes $p_k$ start round $r+1$ with their estimates equal to the decided value $s$.

It then follows from Lemma 4 that, from round $r + 1$, the estimates of all the non-faulty processes keep forever the same value (namely, the decided value). Hence, no value different from this estimate value can be decided. $\square_{Lemma\ 7}$

**Lemma 8.** *Each non-faulty process decides with probability* 1. (Proof in [20].)

**Theorem 6.** *Algorithm* 6 *solves the randomized binary consensus problem in the system model $\mathcal{BZ}\_\mathcal{AS}_{n,t}[t < n/3, \mathrm{VB}, \mathrm{CC}]$.*

**Proof** Follows from lemmas 6, 7 and 8. $\square_{Theorem\ 6}$

**Theorem 7.** *Let $t < n/3$. The expected decision time is constant.*

**Proof** As indicated in the proof of Lemma 8, termination is obtained in two phases. First, all the non-faulty processes adopt the same value $v$. Second, the outcome of the common coin has to be the same as the commonly adopted value $v$.

It follows from the proof of Lemma 8 that there is only one situation in which the non-faulty processes do not adopt the same value. This is when the predicate of line 06 is satisfied for a subset of non-faulty processes and not for the other non-faulty processes. Thus, the expected number of rounds for this to happen is 2. As for the second phase, here again, the probability that the value output by the common coin is the same as the value held by all the non-faulty processes is $1/2$. Thus, the expected time for this to occur is also 2. Combining the two phases, the expected termination time is $4$ rounds (i.e., a small constant).

$\square_{Theorem\ 7}$

# 8 Conclusion

Considering distributed message-passing systems made up of $n$ processes, and where up to $t$ processes may commit Byzantine failures, the aim of the paper was to present in a simple and homogeneous way (a) existing and new broadcast and agreement abstractions, and (b) algorithms implementing them. These broadcast abstractions are UB-broadcast (unreliable broadcast), ND-broadcast (no-duplicity broadcast), RB-broadcast (reliable broadcast), and VB-broadcast (validated broadcast). They have been used to design three multivalued intrusion-tolerant Byzantine consensus algorithms. Moreover, all these algorithms are signature-free. As we have seen, the intrusion-tolerance property means that no value proposed only by Byzantine processes can ever be decided. As a consequence, a default value can be decided when the same value is not proposed by enough processes

The intrusion-tolerant consensus algorithm based on VB-broadcast has several noteworthy features: it is optimal from a resilience point of view ($t < n/3$), each round requires only a single VB-broadcast instance, which costs six communication steps, and the size of control information attached with each message is $O(\log_2 n)$. The paper has also presented a novel randomized binary Byzantine consensus algorithm that is resilient-optimal and, in a very interesting way, is also based on the VB-broadcast abstraction.

# Acknowledgments

The authors want to thank the referees for their constructive comments. This work has been partially supported by the French ANR projects DISPLEXITY and DISCMAT.

# References

[1] Attiya H. and Welch J., *Distributed computing: fundamentals, simulations and advanced topics*, Wiley-Interscience, 414 pages, 2004.

[2] Ben-Or M., Another advantage of free choice: completely asynchronous agreement protocols. *Proc. 2nd ACM Symposium on Principles of Distributed Computing(PODC'83)*, pp. 27-30, 1983.

[3] Bracha G., Asynchronous Byzantine agreement protocols. *Information & Computation*, 75(2):130-143, 1987.

[4] Bracha G. and Toueg S., Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824-840, 1985.

[5] Cachin Ch., Guerraoui R., and Rodrigues L., *Reliable and secure distributed programming*, Springer, 367 pages, 2011.

[6] Cachin Ch., Kursawe K., Petzold F., and Shoup V., Secure and efficient asynchronous broadcast protocols. *Proc. 21st Annual International Cryptology Conference CRYPTO'01*, Springer LNCS 2139, pp. 524-541, 2001.

[7] Cachin Ch., Kursawe K., and Shoup V., Random oracles in Constantinople: practical asynchronous Byzantine agreement using cryptography. *Proc. 19th Annual ACM Symposium on Principles of Distributed Computing (PODC'00)*, pp. 123-132, 2000.
[8] Chandra T. and Toueg S., Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225-267, 1996.
[9] Correia M., Ferreira Neves N., and Verissimo P., From consensus to atomic broadcast: time-free Byzantine-resistant protocols without signatures. *The Computer Journal*, 49(1):82-96, 2006.
[10] Dwork C., Lynch N., and Stockmeyer L., Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2), 288-323, 1988.
[11] Fischer M.J., Lynch N.A., and Paterson M.S., Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374-382, 1985.
[12] Friedman R., Mostéfaoui A., and Raynal M., Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems. *IEEE Transactions on Dependable and Secure Computing*, 2(1):46-56, 2005.
[13] Lamport L., Shostack R., and Pease M., The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3)-382-401, 1982.
[14] Liang G. and Vaidya N., Error-free multi-valued consensus with Byzantine failures. *Proc. 30th ACM Symposium on Principles of Distributed Computing (PODC'11)*, ACM Press, pp. 11-20, 2011.
[15] Lynch N.A., *Distributed algorithms*. Morgan Kaufmann Pub., San Francisco (CA), 872 pages, 1996 (ISBN 1-55860-384-4).
[16] Martin J.-Ph. and Alvisi L., Fast Byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202-215, 2006.
[17] Milosevic Z., Hutle M., and Schiper A., On the reduction of atomic broadcast to consensus with Byzantine faults. *Proc. 30th IEEE Int'l Symp. on Reliable Distributed Systems (SRDS'11)*, pp. 235-244, 2011.
[18] Mostéfaoui A. and Raynal M., Solving consensus using Chandra-Toueg's unreliable failure detectors: a general quorum-based approach. *Proc. 13th Int'l Symposium on Distributed Computing (DISC'99)*, Springer LNCS 1693, pp. 49-63, 1999.
[19] Mostéfaoui A. and Raynal M., Signature-free broadcast-based intrusion tolerance: never decide a Byzantine value. *Proc. 14th Int'l Conference On Principles Of Distributed Systems (OPODIS'10)*, Springer LNCS 6490, pp. 144-159, 2010.
[20] Mostéfaoui A. and Raynal M., Communication and agreement abstractions in the presence of Byzantine processes.*Tech Report #2015*, 24 pages, IRISA, Université de Rennes (France), 2014.
[21] Mostéfaoui A., Raynal M., and Tronel F., From binary consensus to multivalued consensus in asynchronous message-passing systems. *Information Processing Letters*, 73:207-213, 2000.
[22] Pease M., R. Shostak R., and Lamport L., Reaching agreement in the presence of faults. *Journal of the ACM*, 27:228-234, 1980.
[23] Rabin M., Randomized Byzantine generals. *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS'83)*, IEEE Computer Society Press, pp. 116-124, 1983.
[24] Raynal M., *Communication and agreement abstractions for fault-tolerant asynchronous distributed systems*. Morgan & Claypool, 251 pages, 2010 (ISBN 978-1-60845-293-4).
[25] Raynal M., *Fault-tolerant agreement in synchronous message-passing systems*. Morgan & Claypool Publishers, 165 pages, 2010 (ISBN 978-1-60845-525-6).
[26] Raynal M., *Concurrent programming: algorithms, principles and foundations*. Springer, 515 pages, 2013 (ISBN 978-3-642-32026-2).
[27] Srikanth T.K. and Toueg S., Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2:80-94, 1987.
[28] Toueg S., Randomized Byzantine agreement. *Proc. 3rd Annual ACM Symposium on Principles of Distributed Computing (PODC'84)*, pp. 163-178, 1984.
[29] Turpin R. and Coan B.A., Extending binary Byzantine agreement to multivalued Byzantine agreement. *Information Processing Letters*, 18:73-76, 1984.

**Achour Mostéfaoui** is a professor of computer science at the University of Nantes (France). He received his M.Sc. in computer science in 1991, and a Ph.D. in 1994 from the University of Rennes. He is head of a Masters diploma in Computer Science (University of Nantes) and is co-head of the GDD research team within the LINA Lab.

**Michel Raynal** is a professor of computer science at the University of Rennes (France). His main research interests are distributed algorithms, distributed computability, and the fundations of distributed computing. His last two books *Concurrent Programming: Algorithms, Principles and Foundations* (ISBN 978-3-642-32026-2), and *Distributed Algorithms for Message-passing Systems* (ISBN: 978-3-642-38122-5) have been published by Springer in 2013.